CHAPTER 1
ADDING I/O DEVICES TO A MODERN PC

The personal computer (PC) has been around for a long time, (I think that Noah had one on his Ark!). The first IBM PC was announced in 1981, and since then we've all wanted to add hardware to our PCs. The power of the PC has grown—the number of tasks we want the PC to do has grown—and the number of devices we want to connect to the PC has grown. But until recently there has been a practical limit to the number of available ports one could connect devices to.

Enter the Universal Serial Bus, USB. You may have heard USB referred to as "the best thing" to happen to the personal computer for some time. The list of USB features is impressive:

- Hot-pluggable: I/O devices can be added while the PC is running.
- Ease of use: I/O device attachment is recognized by the PC and appropriate device drivers, and configuration is done automatically.
- Single connector type: all devices plug into the same socket type.
- High performance: Up to 480Mbps tranfer rate.
- Three speed choices: match I/O device speed to one of the standard speeds for optimal design.
- Up to 126 devices: there is no practical limit to I/O device expandability.
- Power supplied by cable: most devices will not need an additional power source.
- Power management: devices automatically power down when not in use.
- Error detection and recovery: errors are detected and transactions are retried to ensure that data is delivered reliably.
- External to the PC: there is no need to open the PC or design cards that must be installed in the PC.

Acknowledging the benefits of USB and designing an I/O device interface are two different things, however. Until now, you've had only the USB Specification to read for technical details. A specification by its nature usually provides few if any implementation examples. So, even after reading the specification, you still might not know how to design a simple I/O port for USB, let alone design a telephone or a video-based I/O device.

HANDS-ON EXAMPLES

Most USB books, including the specification, position USB as a technical wonder, which it is. But these books lack practical, how-to examples that include schematics and code. That's where this book comes in. I designed it to help you add I/O devices to your PC, and I have tried to make this as much of a "cookbook" as I can. All the examples are fully documented, and prototype boards are available for you to build upon and expand your ideas and solutions. A variety of vendor solutions are demonstrated so you can choose the solution closest to your application.

I use the term "PC" to refer to Intel-based computers running an operating system that supports USB (I use Windows 98 and Windows 2000 in most of the examples and cover other operating systems in Chapter 16).

- Most of the PC host software examples are implemented in Visual Basic and Visual C++ so that you can choose the language you are most familiar with.
- The microcontroller examples are mainly in MCS51 assembler code but some of the more complex examples are implemented in C; I chose the MCS51 architecture because many manufacturers have used it as the basis for their intelligent USB controllers.
- The 8051 is an uncomplicated architecture with few programming "tricks," so the examples should be easy to port to another microcontroller family.

I do not repeat sections from the USB Specification but have included this on the companion CD-ROM for the interested reader.

This book describes how to connect to a PC host the usual PC-type peripherals, such as scanners, proximity detectors, keypads, and printers, and also how to connect to everyday items such as lights, switches, motors, temperature sensors, speakers, video, and a telephone.

You will soon discover that adding I/O devices to a modern PC host is both easy and fun. The range of devices is limited only by your imagination.

HOW MUCH TECHNICAL BACKGROUND DO YOU NEED?

I assume you have some fundamental electronics and programming skills, but I don't expect these to be your major field. Instead, I assume that you want to **use** the PC to do something useful. This book takes a practical approach to adding devices to a PC. This task is much easier to do today when compared with previous generations of personal computers. Today we connect to an external, standard USB socket and don't even have to open up the case of the computer—a much more civilized approach and far less prone to error.

FOCUS OF THIS BOOK

This book focuses on I/O device design. The book also covers PC host applications and driver software which will let us view and control our I/O devices. There is a lot of USB software on a PC host —we'll see that the implementation of WDM in the Windows family of operating systems uses a layered approach so that we can access USB features and services at a level suitable for our application. One of Intel's goals in working with Microsoft on their USB software was to eliminate the need for most users to write **any** OS-level software; many of the PC host software examples in this book use only applications-level software. When a device driver needs to be written, I cover this in greater detail.

When USB was being developed, a base assumption was the creation of "easy, low-cost I/O devices." The resulting standard is asymmetric with most of the complexity on the PC host side. The operating system and available host controller silicon take care of this complexity so we need deal only with the so-called "easy part," the I/O device. Hub design is also presented, but this is mainly from an I/O device point of view.

THE MODERN PC: A SHORT HISTORY

What is a modern PC? And if you have an 'old' PC, how do you make it modern? This section gives a brief history of the modern PC and introduces USB as a solution to simple, low-cost I/O expansion.

When IBM announced the PC, the company documented the PC internals in their typical thorough style. The availability of this information, along with a strong desire to add hardware to the PC, caused many people to develop plug-in cards so the PC could monitor and even control their environment. Initially the slots inside the IBM PC didn't even have a name. The name ISA, for Industry Standard Architecture, was coined in the early 1980s. There were no official guidelines on how a PC should be expanded, so cards made by vendor A could not be used at the same time as cards made by vendor B. It took until the early 1990s for the ISA bus to be documented (*ISA & EISA Theory and Operation* by Edward Solari is the classic text for this information). Some developers took a lower-risk approach and created attachments that plugged into the serial and parallel ports. But this port resource was quickly depleted, and when more serial and parallel ports were added via plug-in cards, the internal design of the PC could not support enough interrupts, DMA channels, or other system resources. Sometimes the interactions were subtle and failed only under certain conditions.

The bandwidth of these data paths in and out of the PC was also quite low. A better solution was required.

Many PC and PC component vendors, including Compaq, Digital Equipment, NCR, IBM, and Intel Corporation, worked together to define a high-bandwidth expansion bus that was eventually called the Peripheral Components Interconnect bus, or PCI. This bus definition included configuration information and controls to alleviate the vendor-to-vendor conflicts of ISA. The PCI bus was included beside the ISA bus inside the PC host. Software support for automatic configuration was added to Windows so that PCI cards became easy to install. This was the start of Plug and Play, an industry-wide initiative that governs the way add-in hardware should identify itself.

PCI was an instant success for high-bandwidth devices but was seen by many to be excessive and complex for the simpler I/O devices. The PCI interface components typically cost more than a simple I/O device. Another main disadvantage was the location of the PCI connectors; just like their ISA predecessor, they were inside the PC case, which had to be opened, clamping screws undone, boards plugged exactly right into connectors that were difficult to identify, and then the system reassembled. This was discouraging for many would-be users because there was a perceived concern that the PC could be easily broken by these actions. PCI was a better solution but did not address "easy expansion of simple I/O."

Several PC-industry leaders worked together to define an external expansion bus. Driven by customer demand, it was essential that this bus be simple and low cost. Consumer focus groups demanded that PC expansion be as easy as connecting a VCR to a TV—you should not need to set switches or run software, and you must be allowed to plug and swap cables while the equipment is turned on.

Simple to Use and Low Cost

With the goals of simple-to-use and low cost to implement, industry leaders turned their attention to a high-speed serial bus that was later called Universal Serial Bus (USB). Serial was preferred over parallel because the cables would be cheaper, and it would be easier to implement dynamic configuration. "Dynamic configuration" means that the I/O subsystem can be extended or reconfigured by swapping cables while the PC is running. Rebooting a modern PC takes several minutes, so this situation has to be avoided in every solution.

A typical configuration includes one PC host and many I/O devices. To reduce the overall system costs, a master-slave implementation was chosen for USB. The PC host would be the master controlling all traffic on the serial bus—the additional silicon complexity needed to implement the controlling protocols would need to be implemented only once in the host. The slave I/O device could then be made simpler and therefore cheaper. This asymmetric solution means that the two ends of a cable are NOT equal—we need to know which end connects to the master and which end to the slave! The terminology adopted by the USB specification is "upstream" (toward the PC host) and "downstream" (toward the I/O device) (Figure 1-1). The upstream end of the cable controls the protocol and instructs the downstream end to reply at defined times.

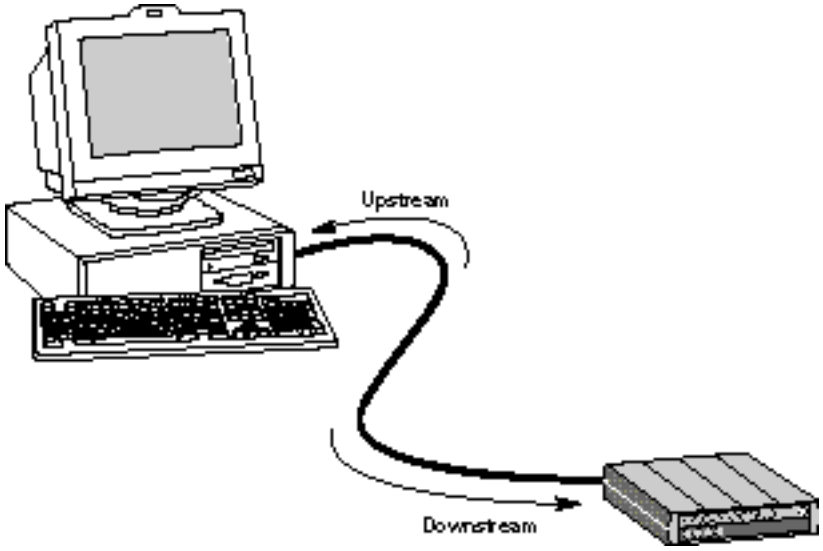


Figure 1-1. A USB cable has two different ends

Another major decision that would support both design goals was to officially supply power from the PC host. Different operating modes that specify power limits are defined, and a good understanding of this feature enables cheaper and easier-to-use I/O devices to be built. You can, for example, eliminate the cost of a power supply in simpler I/O devices and ease the design of more complex I/O devices. Some implementation trade-offs in the I/O devices could also be made when the system specifies minimum and maximum power levels.

USB TERMINOLOGY

The USB specification introduced new terms that are used throughout the USB literature. This section introduces those terms and presents an overview. Later chapters discuss each element in detail.

A typical configuration has a single PC host with multiple **devices** interconnected with USB **cables** (Figure 1-2). The PC host has an embedded **hub**, also called the root hub, which typically contains two or more USB ports.

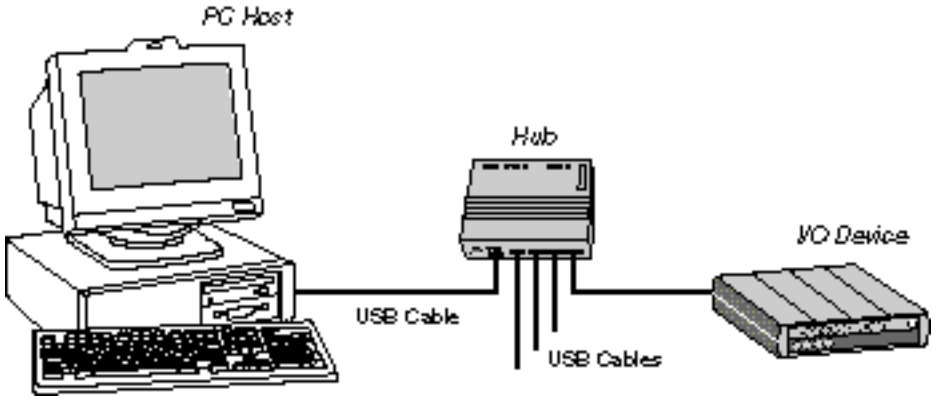
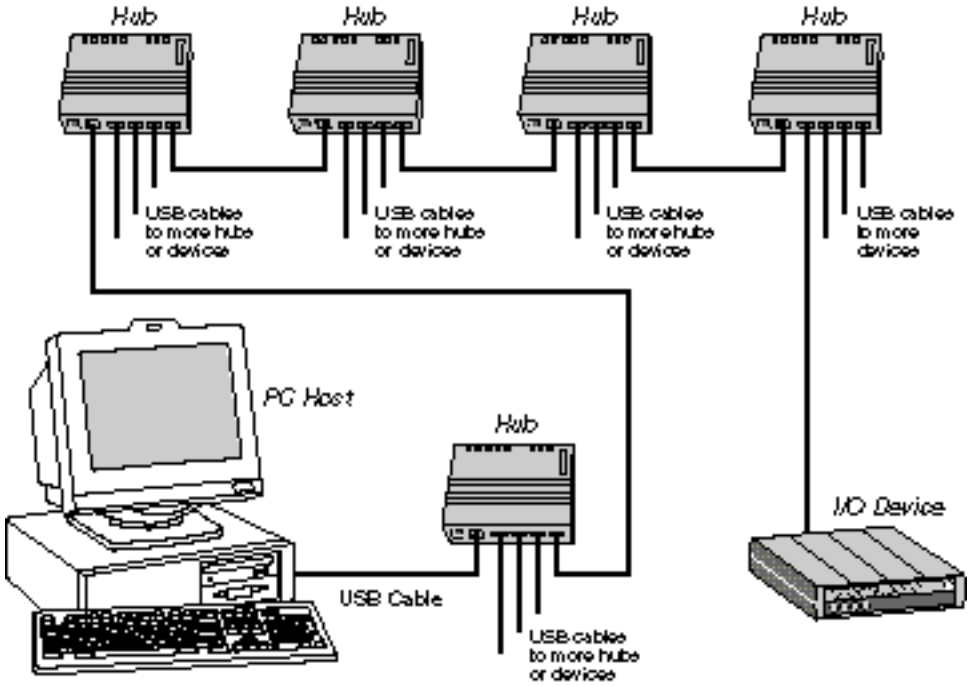


Figure 1-2. Standard USB terminology

Device configurations range from simple to complex:

- Hub: If a device contains only more USB ports, then it is called simply a hub.
- I/O device: An I/O device adds a capability to the PC host. It has a single upstream connection and interacts with the real world to create or consume data on behalf of the PC host.
- Compound device: If a device includes some I/O functionality as well as hub functionality, it is called a compound device (for example, a keyboard that includes additional USB downstream ports).
- Composite device: If a single device implements two or more sets of diverse functions, it is called a composite device (for example, a telephone with keypad and dual audio channels).

As far as the PC host is concerned, devices are the important feature, and up to 126 can be interconnected using external hubs up to five levels deep (Figure 1-3).

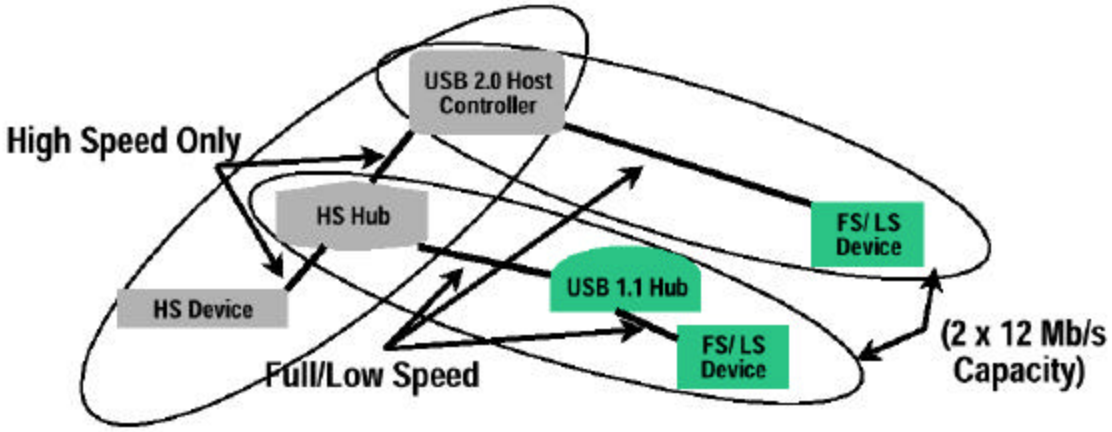


<<new “vertical” diagram coming>>

Figure 1-3. Devices can be nested five hub levels deep

USB now supports three speeds for a device. The USB specification initially defined low speed at 1.5Mbps and full speed at 12Mbps. A new high speed at 480Mbps was recently added to the Version 2.0 of the USB specification. Low speed devices are the cheapest to manufacture and are adequate for supporting low data rate devices such as mice, keyboards and a wealth of other equipment designed to interact with people. The addition of a high-speed data rate enables high bandwidth devices such as full color page scanners and printers and mass storage devices to be designed and added to a PC supporting a high-speed hub.

We shall see that a high-speed (HS) hub is more complex than a full-speed (FS) hub and it is anticipated that it will cost a little more initially. A typical system, therefore, will have a mixture of HS hubs and FS hubs as shown in Figure 1-4.



<< Figure 5-6 from USB 2.0 spec. Need to improve this! >>

Figure 1-4. A typical system using two hub types

Figure 1-4 shows two cable types for illustrative purposes only – in reality all of the USB cables look and operate the same. Note that the HS hubs communicate with each other at 480 Mbps – they also communicate with HS devices at 480Mbps. A FS hub can be connected to a HS hub but it will only communicate at 12 Mbps. However, multiple FS hubs may be attached to a HS hub and each gets their own 12 Mbps channel – the bandwidth is not shared at this level. If a FS hub is connected to another FS hub then the 12 Mbps channel is shared as defined by the original USB specification.

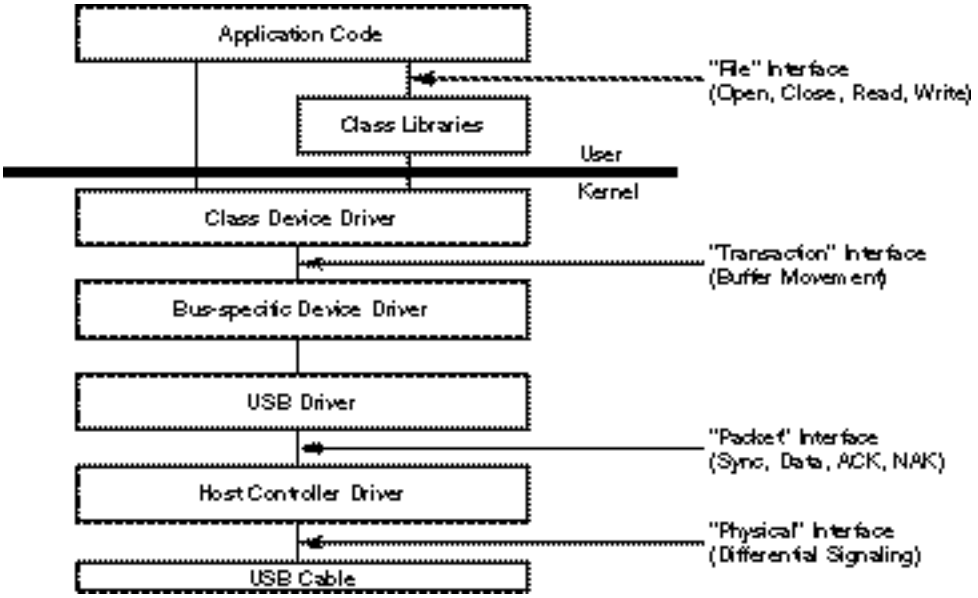
Any speed of I/O device may be connected to any downstream port of any hub. The speed of the I/O device will determine the data rate on the cable to the hub. The only poor choice is attaching a HS device to a FS hub since this will limit the speed of the HS device to 12 Mbps. Fortunately the operating system will recognize this sub-optimal connection during enumeration of the device and will recommend you detach the HS device and re-attach it to a HS hub.

PC Host

A typical configuration has a single PC host. You can interconnect two PC hosts using USB, and this special case is discussed in Chapter 10. The PC host runs a USB-aware operating system software that supports two distinct functions—initialization and run time.

The USB initialization software is active at all times and not just during PC host power-on. Because the initialization software is always active, USB devices can be added and removed at any time. Once a device is added to a PC host, the device is enumerated by the USB initialization software and assigned a unique identifier that is used during run-time. This **enumeration** process is described in detail in Chapter 3.

Figure 1-5 shows how the USB host software is layered; layering supports many different software solutions. A **class** is a grouping of devices, with similar characteristics, that can be controlled by a generic class device driver. Examples of classes include mass-storage devices, communications devices, audio devices, and human interface devices. A single I/O device can belong to multiple classes. If a device fits neatly into one or more of these predefined classes, then you don't need to write operating system software, such as a device driver. The software structure allows you to focus more of your I/O device design efforts on the function and usability of the I/O device and less on the inner workings of an operating system. Vendor-defined commands are also available for those who want specific functionality for their device only.



<<Replace with the correct WDM layered picture>>

Figure 1-5. PC host software for USB is defined in layers

USB Cable

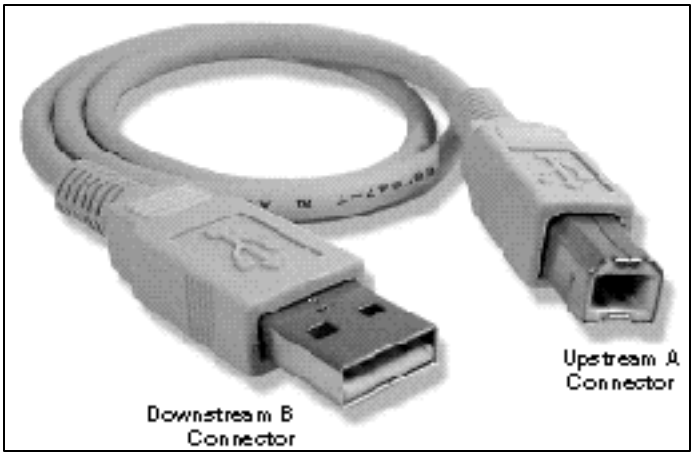
A USB cable includes both power supply and data signals.

Power supplied by the USB cable is an important benefit of the USB specification. A simpler I/O device can rely on the USB cable for all its power needs and will not require the traditional "black brick" plugged into the wall. The power resource is carefully managed by USB with the hub device playing the major role.

A hub or I/O device can be self-powered or bus-powered.

- A bus-powered device relies solely on the USB cable for its power needs and is often a lower-cost alternative.
- Self-powered is the traditional approach in which the hub or I/O device has an additional power cable attached to it.

The USB cable connectors were specially designed with the power pins longer than the signal pins so that power is always applied before signals. Two different connector types were defined to ensure that illegal configurations could not be made (Figure 1-6). An A-type connector defines the downstream end of the cable, and a B-type connector defines the upstream end of the cable.



<< cable ends are mis-labelled >>

Courtesy of Newnex Technology Corp.

Figure 1-6. Different connectors define the ends of a USB cable

The design center for a USB application is a “desktop PC external expansion bus” so a length of 5 meters was chosen as the maximum cable length between a hub and a devices. With 5 levels of hubs, this means that any device will be located within a 30 meter radius of the PC Host. Many applications require a greater distance and the industry has responded with “repeater/extender” products and these are covered in Chapter 9.

A cable with a 480Mbps or even a 12Mbps data rate makes a good antenna, unfortunately, so cable shielding is required. But shielding adds to the cost of the cable. Shielding is not required for the 1.5Mbps signaling rate so this approach is cheaper. The USB specification allows a system to use a mixture of 480Mbps, 12Mbps and 1.5Mbps devices on two cable types (shielded and unshielded) but I recommend that you **only use shielded cables**.

The USB specification defines differential signaling on its data wires to reduce the effects of induced system noise. The USB bus is a point-to-point connection that operates in half-duplex mode. There is no “direction” bit which identifies the current transmitter or receiver – this information is embedded in the bus protocol. While this does simplify the implementation it does make it difficult to build a bus repeater or a bus isolation unit (there are other solutions to these requirements and these are presented in Chapter 9). The fundamental element of communication on the USB bus is a **packet,** and defined sequences of packets are used to build a robust communications channel. Chapter 2 describes packet types and communications protocol in detail.

Hub Device

The hub has two major roles: power management and signal distribution.

An external hub has one upstream connection and multiple downstream connections. The USB specification does not limit the number of downstream connections, but seven seems to be the practical limit. The most popular hub size is four.

A hub can be self-powered or bus-powered.

- Self-powered is the recommended approach in which the hub or I/O device has an additional power cable attached to it. If the hub is self-powered, it can make up to 500 mA available for each of its downstream ports.
- A bus-powered device relies solely on the USB cable for its power needs. If the hub is bus-powered, then it has a maximum of 500 mA available. The hub will use 100 mA for itself and have only 100 mA available for each of four downstream USB ports (unless the socket is suspended).

Because of the power limitation, I do not recommend bus-powered hubs except in exceptional situations. A bus-powered hub can support only 100 mA on each of its downstream USB ports. Although this is enough to enumerate all I/O devices, it is typically not enough for most I/O devices to operate. The current version of Windows 98 does not flash lights or make warning sounds if an enumerated device can't operate because of lack of hub power, so the user is left wondering why the newly attached device doesn't work—not a good user experience. This situation can get worse: Nothing prevents a user from adding a second bus-powered hub onto a port of the first bus-powered hub; the second hub uses all of the 100 mA for itself and cannot supply enough power to even enumerate additional I/O devices. This confuses a user even more. A future release of Windows 98, and Windows 2000, will alert the user if a high-power device is attached to a low-power hub and will recommend a better system configuration.

When first attached to a USB socket, an I/O device (or hub) can always expect 100 mA to be available. The device uses this power to operate during the enumeration stage. The device may NOT use more than 100 mA until it is configured; if it does, the power source on the USB socket will be removed and an error sent to the PC host. If the I/O device requires more than 100 mA for run-time operation, it can request up to 500 mA from the hub. If the hub can supply this power, it does so; otherwise, the I/O device is not configured, and an error message is sent to the PC host. If the I/O device requires more than

500 mA for run-time operation, the device must be self-powered and will need a power cord.

It is worth the design effort to reduce your I/O device power to less than 500 mA because this eliminates the need for an external power source.

An I/O device is required to power itself down, or suspend, when there is no activity on the USB bus. From the USB specification definition, “no activity” means no bus signaling for 3 ms. The maximum amount of power allowed to be drawn during a suspend is 0.5 mA; this is a larger design challenge, as we shall see in later chapters.

I/O Device

An I/O device creates data for, or consumes data from, the real world, as shown in Figure 1-8. A scanner is a good example of a data creator, and a printer is a good example of a data consumer.

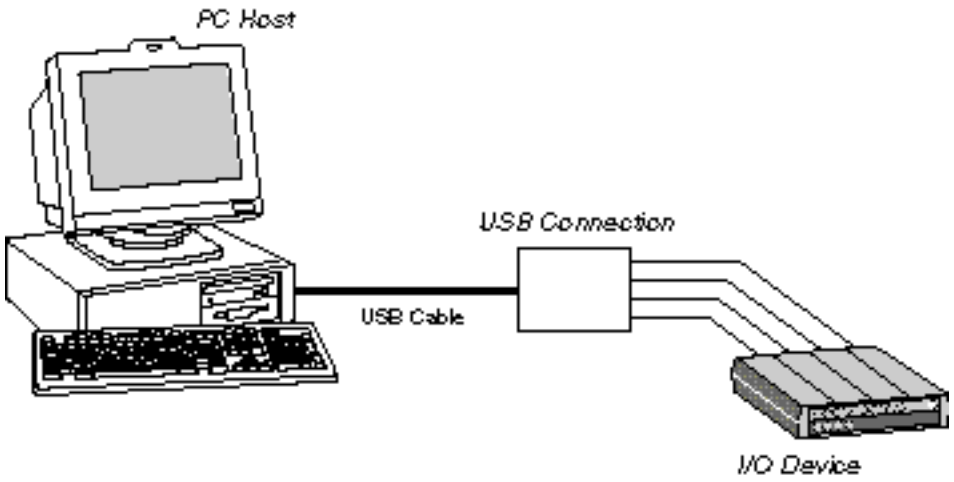
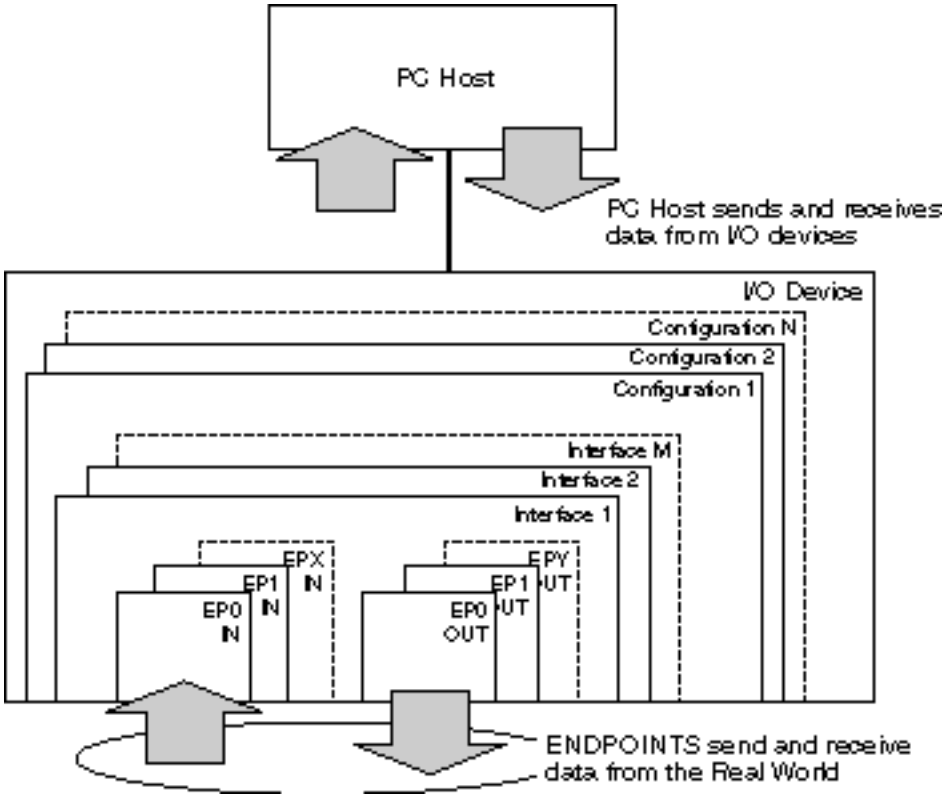


Figure 1-8. I/O device connects USB to the real world

A single I/O device could support both a data creator and a data consumer, so a different software driver may be required for these two distinct tasks. The software, or logical, view of the USB connection is shown in Figure 1-9. The logical view is deliberately general in nature so that all types of real-world connections can be made. This diagram is best explained from the bottom up.

The term **endpoint** is used to describe where data enters or leaves a USB system. An IN endpoint is a data creator, and an OUT endpoint is a data consumer. Note that the data direction is relative to the PC host – if you remember that the PC host is the MASTER controlling all data movements then the data direction is easy to memorize.

A typical real-world connection may need multiple IN and/or OUT endpoints to implement a reliable data delivery scheme. This collection of endpoints is called an **interface** and is directly related to a real-world connection. The operating system will have a software driver that corresponds to each interface. The operating system uses the term **pipe** to describe the logical connection between a software driver on the PC host and the interface on the I/O device. There is always a one-to-one mapping of software driver pipe and interface.



<< add pipes and s/w drivers to this diagram >>

Figure 1-9. Logical view of an I/O device

Some real-world devices may have multiple interfaces—a telephone, for example, has a keypad interface and an audio interface. The operating system will manage the keypad and audio using two separate device drivers. This decomposition of complex devices into smaller logical interfaces means that building-block software elements can be quickly used to manage these complex devices. We don't have to invest the time and effort to write a special telephone device driver—we can be operational with the class drivers already included in the operating system. All the interfaces run concurrently.

A collection of interfaces is called a **configuration,** and only one configuration can be active at a time. A configuration defines the attributes and features of a specific model. Using configurations allows a single USB connection to serve many different roles, and the modularity of this system solution saves development time and support costs.

ADDING USB TO AN "OLD " PC

All desktop PCs and laptops manufactured today contain a USB host controller with one or two available downstream ports. If you have an older PC or laptop it may be easily upgraded with a USB add-in card as shown in Figure 1-10. You should also upgrade your operating system software to ensure that you have the best support for USB.

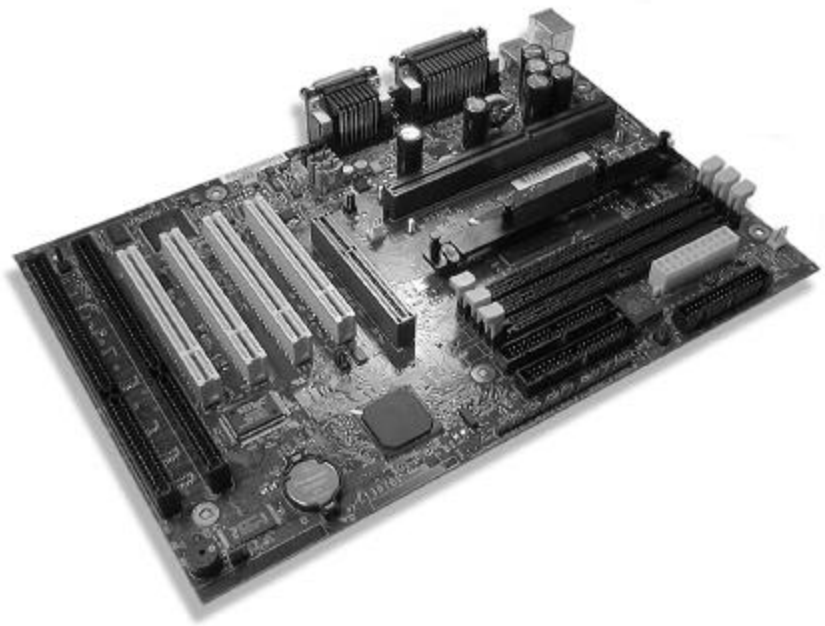
<< Add and describe Lucent's QuadBus card >>

Courtesy of ADS Technologies, Inc. (left) and Entrega Technologies, Inc. (right).

Figure 1-10. Adding USB capability to a PC host

IMPACT OF USB ON PC HOST

As well as simplifying I/O device design, USB can have a major impact on the PC host design. All of the basic features of a PC are implemented in highly integrated components, and a view inside today's PC shows a motherboard similar to Figure 1-11; most of the physical space is taken up by PCI and/or ISA slots!



Courtesy of Systems Group, Intel Corp.

Figure 1-11. Typical PC host motherboard

If all I/O expansion could be external, then there would be no requirement for expansion slots on the motherboard. As well as saving physical space, we also have savings in the power supply design. ISA slots have a power allowance of 25 watts, and PCI slots have a power allowance of 10 watts. This means a PC with no I/O expansion slots ("slotless") needs to have only a 50-to-80-watt power supply rather than a 180- or 240-watt unit. Less heat will be generated inside the system, so a single cooling fan will be adequate. Figure 1-12 shows an example of a slotless motherboard that is only 8 inches square.

Courtesy of Desktop Architecture Laboratory, Intel Corp.

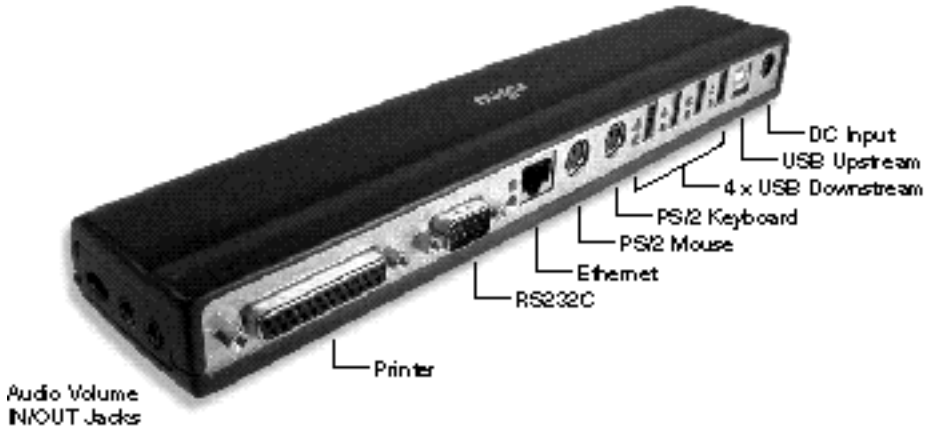
Figure 1-12. Slotless motherboard (8 inches square)

Figure 1-13 shows two concept platforms built around a slotless motherboard. The platforms contain the same motherboard and the same peripherals, a 6-GB hard drive, and a DVD-ROM, and they differ only in the physical placement of the subunits. These PCs are very quiet and don't look like typical "beige-box" PCs. They will be attractive for several new applications such as home use.

Courtesy of Desktop Architecture Laboratory, Intel Corp.

Figure 1-13. Concept PCs built with slotless motherboard

These concept platforms also introduce new I/O paradigms. There are many peripherals available today that attach to the serial or parallel ports of the traditional PC. A peripheral device expander (Figure 1-14) could be used to attach these peripherals to a concept platform. Software drivers on the PC host would redirect I/O requests to this expander, so no changes in applications software would be required. The device shown in Figure 1-14 is from Fujitsu Ltd.; it additionally includes a three-port hub, an Ethernet connection, and sound. Although targeted at USB-enabled laptop computers, the device expander would complement a slotless desktop implementation.



Courtesy of Fujitsu Limited.

<< IBM now has a product like this too, get a photo and permission to use >>

Figure 1-14. Integrated hub and I/O expander products

IMPACT OF USB ON NON-PCS

USB was originally conceived as a desktop PC expansion bus. A large spectrum of devices has been developed which, in turn, is expanding the role of the personal computer. The Windows 95, now 98, operating system was the first to include software support for USB but this too has expanded to include Windows 2000, Linux, MacOS, Windows CE and a variety of other operating systems.

Embedded PC's on industrial buses, such as the example from Radisys shown in Figure 1-15, were also quick to adopt the USB expansion connector. A large array of data acquisition and process control devices, also shown in Figure 1-15, was developed for this industrial application range.

<< Use photos from Radisys, Hohner and National Instruments >>

Figure 1-15. Embedded PC with industrial USB devices

Embedded PC's typically run a real time operating system such as Wind Rivers VxWorks so a variety of vendors now supply USB class and device drivers here too. The volume economics of the PC industry is reducing the price of all USB – related products and this is further expanding USB's application range.

A portable/handheld segment is being fueled by USB. A variety of data collection devices and data storage devices have been developed and newer re-programable devices such as the Journda from Hewlett Packard and the 3650 from Compaq shown in Figure 1-16, include a USB connection to allow data synchomization with a desktop PC. All off these devices are USB slaves and require the PC to implement the data transfer.

A new connector type, the Mini-B, has been approved by the USB Implementors Forum for use in portable devices. This connector, shown in Figure 1-17, is about 12% the size of a standard B connector.

<< Include photographs from Chapter 15 in a collage >>

Figure 1-16. A widening range of portable/handheld USB devices is available

Courtesy of Molex

Figure 1-17. A Mini-B connector for portable devices is available

CHAPTER SUMMARY

The Universal Serial Bus (USB) was the result of a tremendous amount of cooperative industry effort, and its inclusion and operating system support in a PC defines a modern PC. If your old notebook or desktop PC does not have USB sockets, these can be added via a PCMCIA card or a PCI card.

The PC host software is layered and interfaces are defined to allow the simple addition of application programs. The higher up the USB software stack we go, the farther we are from the actual I/O devices we are controlling. This abstraction allows the software and hardware to be developed on different schedules by different people. The hardware and software communicate via standardized interfaces that have the added benefit that they can be either swapped-out or upgraded independently.

Portable/handheld products are now including USB as the standard method of communication – the possibilities for USB-based devices continues to broaden with possibilities now only limited by one's creativity.

My major goal in writing this book was to enable you to quickly take advantage of this ever-increasing opportunity. The rigid USB specification defines exactly how a USB I/O device can correctly inter-operate with a wide spectrum of USB hosts. And this book adds the practical side of what needs to be done to create a compliant USB I/O device.

Read through the examples – they start easy and gradually become more complex. Choose the example closest to what you would like to build and augment from there.

You'll soon discover that USB is both easy and fun.